ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ   
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«СИБИРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ТЕЛЕКОММУНИКАЦИЙ И ИНФОРМАТИКИ»

**Расчетно-графическое задание**

по дисциплине «Защита информации»

«Вариант 1»

|  |  |
| --- | --- |
| Выполнил студент | Свириденко Валентин Вячеславович |
|  |  |

|  |  |
| --- | --- |
| Группы | ИС-741 |
|  |  |

|  |  |
| --- | --- |
| Работу принял | Ассистент кафедры ПмиК Петухова Яна Владимировна |
|  |  |

Новосибирск – 2020

**Содержание**

[**1. Постановка задачи 3**](#_Toc59528073)

[**2. Теоретические сведения 4**](#_Toc59528074)

[**3. Описание разработанного алгоритма. 6**](#_Toc59528075)

[**4. Демонстрация работы программы 8**](#_Toc59528076)

[**СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 9**](#_Toc59528077)

[**ПРИЛОЖЕНИЕ А. Исходный код программы 10**](#_Toc59528078)

1. Постановка задачи

Реализовать алгоритм протокола доказательства с нулевым знанием для задачи «Раскраска графа».

Задача является NP-полной и не имеет быстрых методов для решения, поэтому для тестирования необходимо будет использовать готовые правильные решения. Исходные данные необходимо хранить в файле следующего содержания:

1) в первой строке файла два числа: n - количество вершин графа и m - количество ребер соответственно;

2) в последующих m строках содержится информация о ребрах графа, каждое из которых описывается с помощью двух чисел (номера вершин, соединяемых этим ребром);

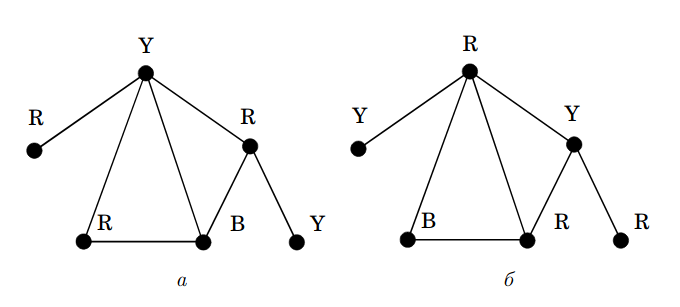
3) в последней строке перечисляются цвета вершин графа по порядку.

Программа должна наглядно демонстрировать работу алгоритма, возможно (но не обязательно) в графическом режиме. Текст программы должен содержать исчерпывающие комментарии, тем не менее, следует воздержаться от описания очевидных действий.

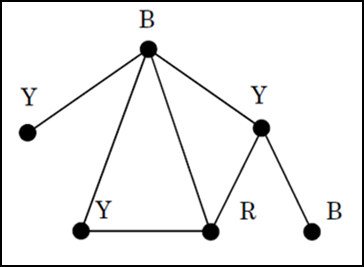
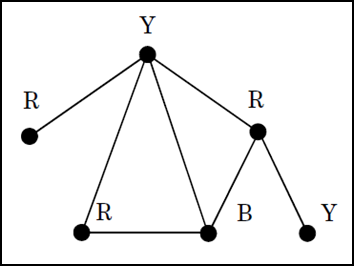
1. Теоретические сведения

Доказательство с нулевым знанием в криптографии – интерактивный криптографический протокол, позволяющий одной из взаимодействующих сторон убедиться в достоверности какого-либо утверждения, не имея при этом никакой другой информации от второй стороны.

В задаче о раскраске графа рассматривается граф с множеством вершин *V* и множеством ребер *E* (числа элементов в этих множествах будем обозначать через |*V*| и |*E*|). Алиса знает правильную раскраску этого графа тремя красками (красной (*R*), синей (*B*) и желтой (*Y*)). Правильная раскраска это такая, когда любые две вершины, соединенные одним ребром, окрашены разными цветами. Приведем пример (Рисунок 2.1).

Рисунок 2.1 – а) правильная раскраска; б) неправильная раскраска

Алиса знает правильную раскраску графа с большими |*V*| и |*E*|. Она хочет доказать это Бобу, но так, чтобы он ничего не узнал об этой раскраске. Протокол доказательства состоит из множества одинаковых этапов. Опишем сначала один этап.

**Шаг 1**. Алиса выбирает случайную перестановку имеющихся цветов и перекрашивает граф. Это не меняет корректность графа, так смежность вешин не меняется.  
  
   
 Рис.2.2 Исходный граф Рис.2.3 Граф после перестановки

**Шаг 2**. Для каждой вершины *v* из множества *V* Алиса генерирует большое случайное число *rv* и заменяет в нем последние биты на те, что соответствуют коду цвета вершины *v*.

**Шаг 3**. Алиса для каждой вершины графа формирует данные из системы RSA:

, и . (1)

**Шаг 4**. Алиса вычисляет для каждой вершины *v*

(2)

и посылает Бобу значения *Nv*, *dv* и *Zv*. для каждой вершины графа.

**Шаг 5**. Боб выбирает случайно одно ребро *ab* из множества *E* и сообщает Алисе, какое именно ребро он выбрал. В ответ Алиса высылает числа *ca* и *cb*, соответствующие вершинам этого ребра. После этого Боб вычисляет

и (3)

и сравнивает младшие биты в полученных числах. При правильной раскраске они должны быть различны. Если значения совпали, значит, Алиса пыталась обмануть Боба.

Описанный алгоритм повторяется *a*|*E*| раз, где *a* > 0 – параметр. Точность работы алгоритма зависит от *a*.

1. Описание разработанного алгоритма.

Алгоритм разработан на языке *Kotlin*.

На рисунке 3.1 – текстовое представление графа, читаемое приложением.



Рисунок 3.1 – Содержимое файла с исходными данными

Граф представляется классом *Graph*.

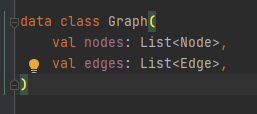


Рисунок 3.2 – Класс *Graph*

В классе содержится список всех его вершин и ребер. Вершина графа в свою очередь представлена классом *Node*.

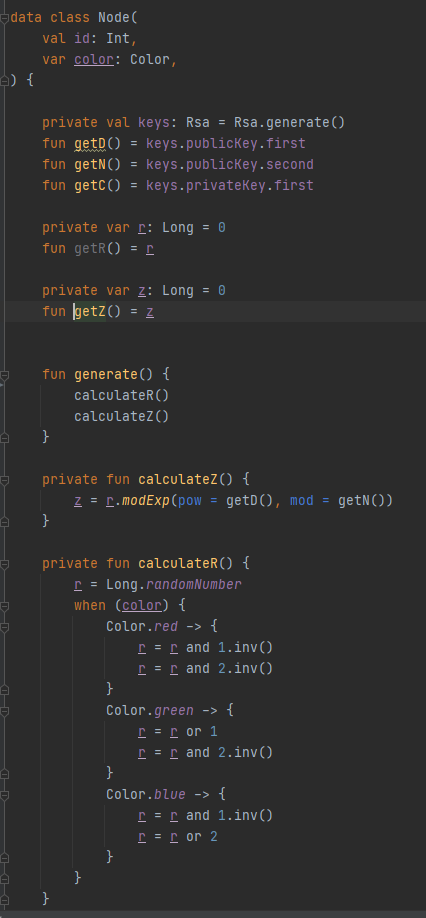


Рисунок 3.3 – Класс *Node*

В этом классе есть ключи, сгенерированные алгоритмом RSA, цвет и дополнительные поля.

Во время чтения файла и парсинга в объекты класса *Graph* считывается информация всех вершин и информация о всех ребрах.

Пример вывода, сформированного программой графа представлен на рисунке 3.4.



Рисунок 3.4 — Вывод сформированного графа

Алгоритм доказательства с нулевым знанием представлен методом *processing* в *GraphRepositoryImpl*. В данном методе реализована работа с клиентом (Алиса) и сервером (Боб).

1. Демонстрация работы программы

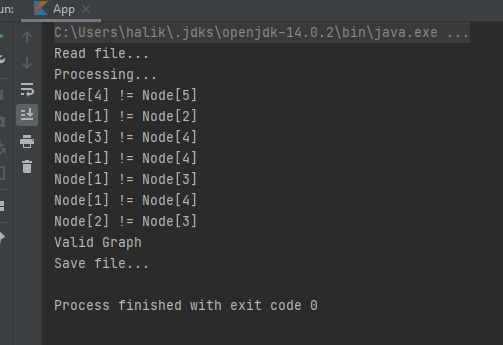


Рисунок 4.1. Вывод программы

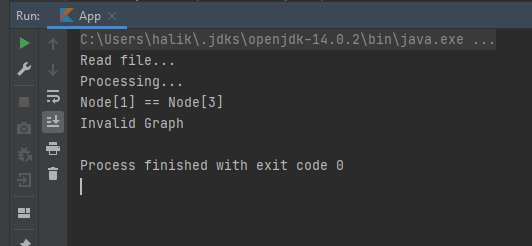


Рисунок 4.2. Демонстрация неудачного выполнения программы

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Рябко, Б. Я. Фионов А. Н. Криптографические методы защиты информации: Учебное пособие для высших учебных заведений. – 2-е издание стереотипное. – Москва: Издательство «Горячаялиния−Телеком», 2012

2. Доказательство с нулевым разглашением [Электронный ресурс]: Википедия – свободная энциклопедия. URL: https://ru.wikipedia.org/wiki/Доказательство\_с\_нулевым\_разглашением (дата обращения: 11.12.2020)

3. Раскраска графов [Электронный ресурс]: Википедия – свободная энциклопедия. URL: https://ru.wikipedia.org/wiki/Раскраска\_графов (дата обращения: 11.12.2020)

ПРИЛОЖЕНИЕ А. Исходный код программы

|  |
| --- |
| graph/src/main/kotlin/me/haliksar/securityalgorithms/graph/App.kt |
| package me.haliksar.securityalgorithms.graph  import kotlinx.coroutines.runBlocking  import me.haliksar.securityalgorithms.graph.di.GraphModules  import me.haliksar.securityalgorithms.graph.domain.usecase.GenerateGraphUseCase  import me.haliksar.securityalgorithms.graph.domain.usecase.GetGraphUseCase  import me.haliksar.securityalgorithms.graph.domain.usecase.ProcessingGraphUseCase  import me.haliksar.securityalgorithms.graph.domain.usecase.SaveGraphUseCase  import org.koin.core.component.KoinComponent  import org.koin.core.component.get  import org.koin.core.context.startKoin  class App : KoinComponent {  companion object {  // graph/src/main/resources/graph\_data.json graph/src/main/resources/graph\_out.json  @JvmStatic  fun main(args: Array<String>) = runBlocking {  initKoin()  App().launch(args)  }  private fun initKoin() {  startKoin {  modules(GraphModules)  }  }  }  suspend fun launch(args: Array<String>) {  when {  args.isEmpty() -> generateStrategy()  args.size == 1 -> fromFileStrategy(args[0])  args.size == 2 -> fromFilesStrategy(args[0], args[1])  }  }  private suspend fun generateStrategy() {  println("Generate...")  val graph = get<GenerateGraphUseCase>().invoke(10000)  get<SaveGraphUseCase>().invoke(graph, "graph/src/main/resources/graph\_save.json")  println("Processing...")  get<ProcessingGraphUseCase>().invoke(graph)  }  private suspend fun fromFileStrategy(filePath: String) {  println("Read file...")  val graph = get<GetGraphUseCase>().invoke(filePath)  println("Processing...")  get<ProcessingGraphUseCase>().invoke(graph)  }  private suspend fun fromFilesStrategy(input: String, output: String) {  println("Read file...")  val graph = get<GetGraphUseCase>().invoke(input)  println("Processing...")  if (get<ProcessingGraphUseCase>().invoke(graph)){  println("Save file...")  get<SaveGraphUseCase>().invoke(graph, output)  }  }  } |
| graph/src/main/kotlin/me/haliksar/securityalgorithms/graph/domain/usecase/SaveGraphUseCase.kt |
| package me.haliksar.securityalgorithms.graph.domain.usecase  import me.haliksar.securityalgorithms.graph.domain.entity.Graph  import me.haliksar.securityalgorithms.graph.domain.repository.GraphRepository  class SaveGraphUseCase(  private val repository: GraphRepository,  ) {  suspend operator fun invoke(graph: Graph, filePath: String) {  repository.saveFromFile(graph, filePath)  }  } |
| graph/src/main/kotlin/me/haliksar/securityalgorithms/graph/domain/usecase/ProcessingGraphUseCase.kt |
| package me.haliksar.securityalgorithms.graph.domain.usecase  import me.haliksar.securityalgorithms.graph.domain.entity.Graph  import me.haliksar.securityalgorithms.graph.domain.repository.GraphRepository  class ProcessingGraphUseCase(  private val repository: GraphRepository,  ) {  suspend operator fun invoke(graph: Graph) {  repository.processing(graph)  }  } |
| graph/src/main/kotlin/me/haliksar/securityalgorithms/graph/domain/usecase/GetGraphUseCase.kt |
| package me.haliksar.securityalgorithms.graph.domain.usecase  import me.haliksar.securityalgorithms.graph.domain.entity.Graph  import me.haliksar.securityalgorithms.graph.domain.repository.GraphRepository  class GetGraphUseCase(  private val repository: GraphRepository,  ) {  suspend operator fun invoke(filePath: String): Graph =  repository.getFromFile(filePath)  } |
| graph/src/main/kotlin/me/haliksar/securityalgorithms/graph/domain/usecase/GenerateGraphUseCase.kt |
| package me.haliksar.securityalgorithms.graph.domain.usecase  import me.haliksar.securityalgorithms.graph.domain.entity.Graph  import me.haliksar.securityalgorithms.graph.domain.repository.GraphRepository  class GenerateGraphUseCase(  private val repository: GraphRepository,  ) {  suspend operator fun invoke(countNodes: Int): Graph =  repository.generate(countNodes)  } |
| graph/src/main/kotlin/me/haliksar/securityalgorithms/graph/domain/repository/GraphRepository.kt |
| package me.haliksar.securityalgorithms.graph.domain.repository  import me.haliksar.securityalgorithms.graph.domain.entity.Graph  interface GraphRepository {  suspend fun getFromFile(filePath: String): Graph  suspend fun generate(countNodes: Int): Graph  suspend fun saveFromFile(graph: Graph, filePath: String)  suspend fun show(graph: Graph)  suspend fun processing(graph: Graph): Boolean  } |
| graph/src/main/kotlin/me/haliksar/securityalgorithms/graph/domain/entity/Node.kt |
| package me.haliksar.securityalgorithms.graph.domain.entity  import me.haliksar.securityalgorithms.graph.data.generator.Rsa  import me.haliksar.securityalgorithms.libs.core.prime.randomNumber  import me.haliksar.securityalgorithms.libs.modexp.modExp  data class Node(  val id: Int,  var color: Color,  ) {  private val keys: Rsa = Rsa.generate()  fun getD() = keys.publicKey.first  fun getN() = keys.publicKey.second  fun getC() = keys.privateKey.first  private var r: Long = 0  fun getR() = r  private var z: Long = 0  fun getZ() = z  fun generate() {  calculateR()  calculateZ()  }  private fun calculateZ() {  z = r.modExp(pow = getD(), mod = getN())  }  private fun calculateR() {  r = Long.randomNumber  when (color) {  Color.red -> {  r = r and 1.inv()  r = r and 2.inv()  }  Color.green -> {  r = r or 1  r = r and 2.inv()  }  Color.blue -> {  r = r and 1.inv()  r = r or 2  }  }  }  } |
| graph/src/main/kotlin/me/haliksar/securityalgorithms/graph/domain/entity/Graph.kt |
| package me.haliksar.securityalgorithms.graph.domain.entity  data class Graph(  val nodes: List<Node>,  val edges: List<Edge>,  ) |
| graph/src/main/kotlin/me/haliksar/securityalgorithms/graph/domain/entity/Edge.kt |
| package me.haliksar.securityalgorithms.graph.domain.entity  data class Edge(  val nodeId1: Int,  val nodeId2: Int,  ) |
| graph/src/main/kotlin/me/haliksar/securityalgorithms/graph/domain/entity/Color.kt |
| package me.haliksar.securityalgorithms.graph.domain.entity  enum class Color { red, green, blue } |
| graph/src/main/kotlin/me/haliksar/securityalgorithms/graph/di/GraphModules.kt |
| package me.haliksar.securityalgorithms.graph.di  import com.beust.klaxon.Klaxon  import me.haliksar.securityalgorithms.graph.data.client.Client  import me.haliksar.securityalgorithms.graph.data.client.ClientImpl  import me.haliksar.securityalgorithms.graph.data.datasource.GraphLocalDataSource  import me.haliksar.securityalgorithms.graph.data.datasource.GraphLocalDataSourceImpl  import me.haliksar.securityalgorithms.graph.data.file\_manager.FileManager  import me.haliksar.securityalgorithms.graph.data.file\_manager.JsonGraphManagerImpl  import me.haliksar.securityalgorithms.graph.data.generator.GraphGenerator  import me.haliksar.securityalgorithms.graph.data.generator.GraphGeneratorImpl  import me.haliksar.securityalgorithms.graph.data.repository.GraphRepositoryImpl  import me.haliksar.securityalgorithms.graph.data.server.Server  import me.haliksar.securityalgorithms.graph.data.server.ServerImpl  import me.haliksar.securityalgorithms.graph.domain.entity.Graph  import me.haliksar.securityalgorithms.graph.domain.repository.GraphRepository  import me.haliksar.securityalgorithms.graph.domain.usecase.GenerateGraphUseCase  import me.haliksar.securityalgorithms.graph.domain.usecase.GetGraphUseCase  import me.haliksar.securityalgorithms.graph.domain.usecase.ProcessingGraphUseCase  import me.haliksar.securityalgorithms.graph.domain.usecase.SaveGraphUseCase  import org.koin.dsl.module  private val generatorsModule = module {  single<GraphGenerator> { GraphGeneratorImpl() }  }  private val parserModule = module {  single { Klaxon() }  }  private val fileManagerModule = module {  single<FileManager<Graph>> { JsonGraphManagerImpl(klaxon = get()) }  }  private val dataSourcesModule = module {  single<GraphLocalDataSource> { GraphLocalDataSourceImpl(fileManager = get(), generator = get()) }  }  private val repositoriesModule = module {  single<GraphRepository> {  GraphRepositoryImpl(  dataSource = get(),  server = get(),  client = get(),  )  }  }  private val useCasesModule = module {  single { GenerateGraphUseCase(get()) }  single { GetGraphUseCase(get()) }  single { SaveGraphUseCase(get()) }  single { ProcessingGraphUseCase(get()) }  }  private val clientModule = module {  single<Client> { ClientImpl() }  }  private val serverModule = module {  single<Server> { ServerImpl() }  }  val GraphModules = listOf(  parserModule,  generatorsModule,  fileManagerModule,  dataSourcesModule,  repositoriesModule,  useCasesModule,  clientModule,  serverModule,  ) |
| graph/src/main/kotlin/me/haliksar/securityalgorithms/graph/data/server/Server.kt |
| package me.haliksar.securityalgorithms.graph.data.server  import me.haliksar.securityalgorithms.graph.domain.entity.Edge  import me.haliksar.securityalgorithms.graph.domain.entity.Graph  import me.haliksar.securityalgorithms.libs.modexp.modExp  interface Server {  fun chooseEdge(graph: Graph): Edge  fun check(graph: Graph, current: Edge): Boolean  }  class ServerImpl : Server {  override fun chooseEdge(graph: Graph): Edge = graph.edges.random()  override fun check(graph: Graph, current: Edge): Boolean {  val node1 = graph.nodes[current.nodeId1]  val node2 = graph.nodes[current.nodeId2]  val val1 = node1.getZ().modExp(node1.getC(), node1.getN()) and 7  val val2 = node2.getZ().modExp(node2.getC(), node2.getN()) and 7  if (val1 != val2) {  println("Node[${current.nodeId1}] != Node[${current.nodeId2}]")  } else {  println("Node[${current.nodeId1}] == Node[${current.nodeId2}]")  }  return val1 == val2  }  } |
| graph/src/main/kotlin/me/haliksar/securityalgorithms/graph/data/repository/GraphRepositoryImpl.kt |
| package me.haliksar.securityalgorithms.graph.data.repository  import me.haliksar.securityalgorithms.graph.data.client.Client  import me.haliksar.securityalgorithms.graph.data.datasource.GraphLocalDataSource  import me.haliksar.securityalgorithms.graph.data.server.Server  import me.haliksar.securityalgorithms.graph.domain.entity.Graph  import me.haliksar.securityalgorithms.graph.domain.repository.GraphRepository  class GraphRepositoryImpl(  private val dataSource: GraphLocalDataSource,  private val server: Server,  private val client: Client,  ) : GraphRepository {  override suspend fun getFromFile(filePath: String): Graph =  dataSource.getFromFile(filePath)  override suspend fun generate(countNodes: Int): Graph =  dataSource.generate(countNodes)  override suspend fun saveFromFile(graph: Graph, filePath: String) {  dataSource.saveFromFile(graph, filePath)  }  override suspend fun show(graph: Graph) {  dataSource.show(graph)  }  override suspend fun processing(graph: Graph): Boolean {  repeat(graph.edges.size) {  client.shuffle(graph)  val edge = server.chooseEdge(graph)  if (server.check(graph, edge)) {  return false  }  }  return true  }  } |
| graph/src/main/kotlin/me/haliksar/securityalgorithms/graph/data/generator/Rsa.kt |
| package me.haliksar.securityalgorithms.graph.data.generator  import me.haliksar.securityalgorithms.libs.core.prime.mutuallyPrime  import me.haliksar.securityalgorithms.libs.core.prime.randomPrimeNumber  import me.haliksar.securityalgorithms.libs.gcd.extendedGcdTailRec  data class Rsa(  val publicKey: Pair<Long, Long>,  val privateKey: Pair<Long, Long>,  ) {  companion object {  fun generate(): Rsa {  val p = Long.randomPrimeNumber  val q = Long.randomPrimeNumber  val n = p \* q // модуль  val f = (p - 1L) \* (q - 1L) // функция Эйлера  val d = Long.mutuallyPrime(f) // открытая экспонента, простая из чисел Ферма  var c = d.extendedGcdTailRec(f).y // Секретная экспонента  if (c < 0) c += f  return Rsa(publicKey = Pair(d, n), privateKey = Pair(c, n))  }  }  } |
| graph/src/main/kotlin/me/haliksar/securityalgorithms/graph/data/generator/GraphGenerator.kt |
| package me.haliksar.securityalgorithms.graph.data.generator  import me.haliksar.securityalgorithms.graph.domain.entity.Color  import me.haliksar.securityalgorithms.graph.domain.entity.Edge  import me.haliksar.securityalgorithms.graph.domain.entity.Graph  import me.haliksar.securityalgorithms.graph.domain.entity.Node  interface GraphGenerator {  suspend fun generate(countNodes: Int): Graph  }  class GraphGeneratorImpl : GraphGenerator {  override suspend fun generate(countNodes: Int): Graph {  val nodes: List<Node> = (0..countNodes).shuffled().map {  Node(it, Color.values().random())  }  val edges = mutableListOf<Edge>()  for (i in 0 until countNodes - 1) {  edges.add(Edge(nodes[i].id, nodes[i + 1].id))  }  return Graph(nodes, edges)  }  } |
| graph/src/main/kotlin/me/haliksar/securityalgorithms/graph/data/file\_manager/FileManager.kt |
| package me.haliksar.securityalgorithms.graph.data.file\_manager  import com.beust.klaxon.Klaxon  import me.haliksar.securityalgorithms.graph.domain.entity.Graph  import java.io.File  interface FileManager<D> {  suspend fun save(data: D, filePath: String)  suspend fun get(filePath: String): D  }  class JsonGraphManagerImpl(  private val klaxon: Klaxon,  ) : FileManager<Graph> {  override suspend fun save(data: Graph, filePath: String) {  val json = klaxon.toJsonString(data)  File(filePath).apply {  createNewFile()  writeText(json)  }  }  override suspend fun get(filePath: String): Graph =  klaxon.parse<Graph>(File(filePath))  ?: throw Exception("Graph invalid")  } |
| graph/src/main/kotlin/me/haliksar/securityalgorithms/graph/data/datasource/GraphLocalDataSource.kt |
| package me.haliksar.securityalgorithms.graph.data.datasource  import kotlinx.coroutines.Dispatchers  import kotlinx.coroutines.withContext  import me.haliksar.securityalgorithms.graph.data.file\_manager.FileManager  import me.haliksar.securityalgorithms.graph.data.generator.GraphGenerator  import me.haliksar.securityalgorithms.graph.domain.entity.Graph  interface GraphLocalDataSource {  suspend fun getFromFile(filePath: String): Graph  suspend fun generate(countNodes: Int): Graph  suspend fun saveFromFile(graph: Graph, filePath: String)  suspend fun show(graph: Graph)  }  class GraphLocalDataSourceImpl(  private val fileManager: FileManager<Graph>,  private val generator: GraphGenerator,  ) : GraphLocalDataSource {  override suspend fun getFromFile(filePath: String): Graph =  withContext(Dispatchers.IO) {  fileManager.get(filePath)  }  override suspend fun saveFromFile(graph: Graph, filePath: String) =  withContext(Dispatchers.IO) {  fileManager.save(graph, filePath)  }  override suspend fun generate(countNodes: Int): Graph =  withContext(Dispatchers.IO) {  generator.generate(countNodes)  }  override suspend fun show(graph: Graph) {  println(graph)  }  } |
| graph/src/main/kotlin/me/haliksar/securityalgorithms/graph/data/client/Client.kt |
| package me.haliksar.securityalgorithms.graph.data.client  import me.haliksar.securityalgorithms.graph.domain.entity.Edge  import me.haliksar.securityalgorithms.graph.domain.entity.Graph  import me.haliksar.securityalgorithms.graph.domain.entity.Node  import me.haliksar.securityalgorithms.libs.modexp.modExp  interface Client {  fun shuffle(graph: Graph)  }  class ClientImpl : Client {  override fun shuffle(graph: Graph) {  val colorList = graph.nodes.map { it.color }.shuffled()  colorList.forEachIndexed { index, color ->  graph.nodes[index].color = color  graph.nodes[index].generate()  }  }  } |